Санкт–Петербургский государственный университет

ГОГОТОВ Александр Сергеевич

Выпускная квалификационная работа

Управление сервером в контуре обратной связи

Уровень образования: бакалавриат
Направление 02.03.02 «Фундаментальная информатика и информационные технологии»
Основная образовательная программа СВ.5003.2016 «Программирование и информационные технологии»
Профиль «Автоматизация научных исследований»

Научный руководитель:
профессор, зав. кафедрой компьютерных технологий
и систем, д.ф.-м.н. Веремей Евгений Игоревич

Рецензент:
доцент кафедры компьютерного моделирования
и многопроцессорных систем,
к.т.н. Гришкин Валерий Михайлович

Санкт-Петербург
2020 г.

Содержание

Введение

В современном мире весьма актуальны разнообразные задачи, связанные с изменением динамических свойств различных объектов, которые решаются с применением практических методов теории управления. Под управлением в широком смысле понимается достижение и поддержание некоторого желаемого уровня выходных сигналов с помощью управляющих входных сигналов. При наличии математической модели динамического объекта, процесса или явления, описывающей преобразование входных сигналов в выходные [1], можно говорить о формализации задачи управления.

На практике используют либо программное управление, либо управление с обратной связью. В первом случае на вход объекта подается сигнал, представляемый заданной функцией времени. Для варианта с обратной связью во входной сигнал вводится в каждый момент времени информация об отклонении измеренных выходных сигналов от желаемых значений. При этом вычисления производятся в непрерывном цикле: измеряется отклонение, определяются управляющие сигналы, эти сигналы подаются на приводы управляющих органов.

Основным достоинством схемы управления с контуром обратной связи является отсутствие необходимости в точном задании заранее не известных внешних возмущений [3], к которым замкнутая система автоматически адаптируется. Особую роль играют вопросы управления компьютерными системами. Для них используются специфические представления о качестве функционирования, в отличие от природных или механических объектов. Здесь применяются логические количественные характеристики, определяющие возможность более свободного выбора метрик, используемых для оценок отклонений от заданных выходных сигналов [6]. Как и для других объектов, представляется весьма актуальным применение управления с обратной связью для компьютерных систем. В частности, к таким системам относятся широко распространенные веб-серверы.

Веб-сервер — это вычислительный комплекс, осуществляющий обработку HTTP-запросов пользователей. Этот комплекс представляет собой сочетание аппаратного и программного обеспечения (ПО). С аппаратной точки

зрения работа веб-сервера характеризуется рядом динамических характеристик, таких как уровень загрузки центрального процессора (ЦП), использование оперативной памяти, скорость обработки запросов, время ответа. С программной точки зрения следует выделить настройки обработки запросов — параметры, доступные оператору в целях задания конфигурации работы сервера. С их помощью могут регулироваться используемые функции, временные и количественные характеристики, непосредственно влияющие на процесс обработки запросов. Следовательно, такие параметры могут быть использованы в качестве входных сигналов при построении управления, а в роли выходных сигналов могут выступать динамические характеристики. Исследования в области моделирования и управления веб-сервером направлены на достижение полностью автоматического регулирования его работы с учетом изменяющихся требований и внешних условий.

Построение модели веб-сервера включает в себя определение ее вида, определение типа зависимости динамических характеристик от конфигурационных параметров, вычисление коэффициентов модели. В конечном счете это позволит обеспечить непрерывное управление динамическими характеристиками аппаратного обеспечения без привязки к программным характеристикам, что представляется универсальным и независимым от особенностей конкретной системы подходом к решению задач, связанных с обслуживанием и обеспечением работоспособности веб-серверов. Реализация такого управления может быть использована как основа для построения решения с применением искусственного интеллекта, что обусловливает важность и современность подобного подхода.

Постановка задачи

Конкретная реализация веб-сервера включает в себя набор настраиваемых (конфигурационных) параметров, предназначенных для его оператора или администратора. Их воздействие непосредственно связано с программным устройством сервера. Предполагается, что такая реализация выбрана и что работа сервера существенно зависит от используемых значений некоторых конфигурационных параметров, называемых управляющими. Задача состоит в том, чтобы провести ряд натурных экспериментов с конткретным сервером для формирования связи его динамических характеристик с управляющими параметрами в форме математической модели.

Мотивацией для построения управления с обратной связью может являться:

- обеспечение отказоустойчивости системы веб-серверов;
- балансировка нагрузки;
- оптимизация режимов работы сервера.

Применение управления в контуре обратной связи для достижения отказоустойчивости системы из нескольких веб-серверов можно проиллюстрировать примером из [3]. Пусть некоторая работа выполняется на трех серверах одновременно. Для того чтобы система сохраняла работоспособность при выходе одного из серверов из строя, необходимо добиться того, чтобы загрузка ЦП каждого сервера не превышала 66% от максимальной. Тогда при отказе одного сервера можно будет увеличить загрузку ЦП оставшихся серверов до 100% с сохранением производительности системы.

В качестве динамической характеристики в данной работе рассматривается мера загрузки ЦП. Роль управлений играют два конфигурационных параметра, наиболее существенно влияющих на загрузку ЦП сервера. В качестве ПО сервера был выбран Apache HTTP Server [20] — популярный и поддерживаемый продукт с открытым исходным кодом. Для аппаратного обеспечения используется неспециализированный компьютер — домашний ноутбук.

Понятие о функционировании сервера имеет смысл только тогда, когда на этот сервер воздействует некоторая рабочая нагрузка, соответствующая

внешним условиям. Для HTTP-сервера она выражается в более или менее регулярных HTTP-запросах, которые клиенты (пользователи) отправляют на сервер. Поскольку в работе используется экспериментальный сервер, важнейшим шагом при подготовке к синтезу управления является создание искусственной нагрузки. Это достигается с помощью нагрузочного тестирования — имитации активности пользователей. Для реализации подобного тестирования применяются так называемые нагрузочные имитаторы — программные продукты и приложения, с помощью которых генерируются виртуальные пользователи.

К настоящему моменту существует большое число нагрузочных имитаторов, многие из которых распространяются бесплатно и с открытым исходным кодом. Таким образом, ключевым вопросом представляется выбор средства нагрузочного тестирования. Чтобы эксперименты можно было проводить с учетом ограничений условий работы и в целях обеспечения достаточной выразительности экспериментов, нагрузочный имитатор должен удовлетворять ряду требований, определяемых практическими ограничениями.

В связи с необходимостью соблюдения чистоты и логичности экспериментов особый вопрос заключается не только в выборе конкретного нагрузочного имитатора, но и в организации тестирования. В целом можно сформулировать следующие требования к нагрузочному тестированию:

- возможность запуска не ограниченных по времени тестов;
- создание достаточно большой нагрузки — обеспечивающей загрузку ЦП сервера вплоть до 100%;
- приближение сценария действий виртуальных пользователей к поведению реальных пользователей;
- исключение возможности блокировки работы сервера со стороны нагрузочного имитатора;
- удовлетворение физических возможностей используемых вычислительных устройств требованиям работы с нагрузочным имитатором.

После решения задачи воссоздания условий работы сервера для построения управления необходимо реализовать программу, которая обеспечит ав-

томатизированное проведение натурных экспериментов. Функциональность программы должна обеспечивать следующие возможности [2]:

– автоматическое варьирование конфигурационных параметров по заданному сценарию;
– измерение значений загрузки ЦП с достаточно малым временным шагом;
– построение модели сервера, исходя из экспериментальных данных.

Таким образом, в данной работе будут рассмотрены следующие вопросы:

– организация нагрузочного тестирования;
– проведение натурных экспериментов с веб-сервером;
– построение и исследование модели сервера как объекта управления.

Сервер является вычислительной системой, работающей с сигналами дискретного времени. Взаимосвязь между такими сигналами естественным образом представляется в виде разностных уравнений [3]. В простейшем варианте математическую модель сервера, получаемую в результате работы, целесообразно искать в виде линейной стационарной системы разностных уравнений [1]. Линейный вид модели представляет из себя существенное упрощение по сравнению со сложным устройством такой системы, как веб-сервер, однако подобные модели хорошо изучены, и с их помощью часто возможно достигнуть приемлемой точности.

Система уравнений динамики сервера n-го порядка с m параметров в общем виде выглядит следующим образом:

$$y(k+1) = f(\boldsymbol{y}, \boldsymbol{u}, \boldsymbol{h}),$$

где $\boldsymbol{y} = (y(k), y(k-1), ..., y(k-n))$ — вектор значений загрузки ЦП сервера в соответствующие моменты дискретного времени, $\boldsymbol{u} = (u_i(k), u_i(k-1), ..., u_i(k-n))$ — матрица значений управляющих параметров в соответствующие моменты дискретного времени ($i = 1, 2, ..., m$ —

номер управляющего параметра), $\boldsymbol{h}$ — вектор параметров модели. В зависимости от режима проведения экспериментов вектор $\boldsymbol{h}$ может включать в себя настройки нагрузочного имитатора, значения конфигурационных параметров сервера, не выбранных в качестве управляющих, настройки аппаратного обеспечения и операционной системы (ОС).

Для данной модели возможно осуществить выбор рабочей точки $(\overline{\boldsymbol{y}}, \overline{\boldsymbol{u}})$, такой что

$$\overline{\boldsymbol{y}} \equiv f(\overline{\boldsymbol{y}}, \overline{\boldsymbol{u}}, \boldsymbol{h}),$$

в любой момент дискретного времени.

В настоящей работе рассматривается простейший вариант, при котором $n = 1$ и $m = 2$. В таком случае можно построить линейное приближение этого уравнения в отклонениях от рабочей точки, которое будет иметь вид:

$$\tilde{y}(k+1) = \alpha_1 \tilde{y}(k) + \beta_1 \tilde{u}_1(k) + \beta_2 \tilde{u}_2(k), \tag{1}$$

где $\tilde{y}(k) = y(k) - \overline{y}(k)$; $\tilde{u}_i(k) = u_i(k) - \overline{u}_i(k)$, $i = 1, 2$; α_1, β_1 и β_2 — коэффициенты уравнения. С помощью данной модели будет возможно построение управления в контуре обратной связи с использованием пропорционального регулятора.

Обзор литературы

Предмет данной работы является достаточно специфическим и слабо изученным к настоящему моменту. Использованные источники в основном представляют собой англоязычные статьи, книги и электронные ресурсы, не затрагивающие всех аспектов поставленной задачи. Идея работы основывается на ряде исследований, сформулированных в книге [3]. В этом источнике управление Apache HTTP Server используется в качестве одного из примеров приложения теории, касающейся управления вычислительными системами в контуре обратной связи. Так, в главе 2 рассматривается построение уравнения динамики сервера, исходя из экспериментальных данных, описывается устройство сервера в одном из вариантов. Основной трудностью использования работы является ее недостаточная актуальность, так как она была опубликована в 2004 году. Более подробное описание достигнутых в главе 2 результатов представлено в работе [2], предложены последовательность действий и подход к программному решению задачи, описаны числовые результаты.

В книге [1] дано теоретическое представление линейных стационарных систем разностных уравнений и их приложений в задачах моделирования цифровых объектов.

В работе [6] приведен обзор принципов и примеров управления компьютерными системами в контуре обратной связи. Исследуются области применения различных регуляторов. Достоинствами данной работы в сравнении с [3] являются больший акцент на практической части исследований, а также ее относительная новизна (2013 год). Однако в этой книге управление серверами рассматривается достаточно поверхностно, применение загрузки ЦП или использования оперативной памяти в качестве выходных сигналов не изучается.

Большую роль в решении поставленной задачи играет вопрос нагрузочного тестирования сервера. В работах [5], [4] рассматриваются методы генерации нагрузки, соответствующей сценариям работы реальных веб-серверов. Статьи [22] и [21] представляют собой всесторонний и актуальный обзор средств нагрузочного тестирования. На основании данных работ возможно выбрать подходящие методы и сценарии условий работы сервера.

Глава 1. Организация нагрузочного тестирования

В настоящей главе рассматриваются вопрос выбора наиболее подходящего нагрузочного имитатора для проведения натурных экспериментов, а также подготовка среды нагрузочного тестирования в целях соответствия требованиям экспериментов.

1.1 Обзор нагрузочных имитаторов

В ходе исследования было изучено более десяти различных средств нагрузочного тестирования [22, 21]. Большинство из них оказались непригодными даже для настройки и проверки. В частности, были выявлены следующие недостатки предложенных программных продуктов:

- программы Tsung [18], Apache Siege, LoadImpact K6 [15] невозможно или затруднительно использовать на устройстве с ОС Windows;
- средства Solex, HTTPerf [14, 5], OpenSTA [17] являются устаревшими;
- Goad [12], Gatling [11], Grinder [13] слишком сложны в изучении;

Поскольку в данной работе исследуется сама возможность управления динамикой сервера в контуре обратной связи, предпочтение отдавалось универсальным и легко изучаемым нагрузочным имитаторам. Три программных продукта были исследованы более подробно.

1. Apache Bench [9] — консольное приложение, осуществляющее генерацию заданного количества виртуальных пользователей, которые параллельно или последовательно направляют по одному HTTP-запросу по заданному IP-адресу. Это средство поставляется вместе с Apache HTTP Server. Нагрузочный имитатор может быть запущен в командной строке ОС Windows следующей командой:

```
ab -n <общее число запросов>
 -c <число параллельно отправляемых запросов>
 -s <время ожидания ответа от сервера, с> -k
 <IP-адрес сервера>
```

После выполнения тестирования пользователю программы предоставляется информация о скорости обработки запросов, объеме переданной информации и распределении времени ответа сервера.

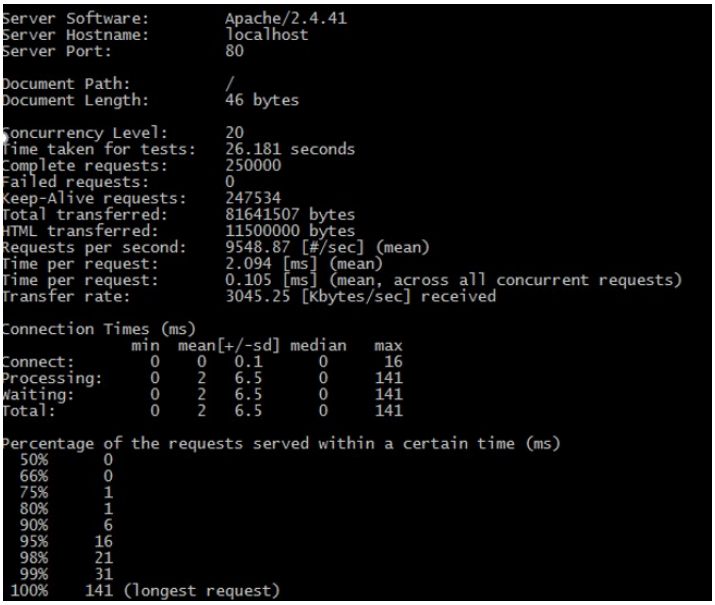

Рис. 1: Пример вывода работы программы Apache Bench.

Из представленного описания можно сделать вывод, что этот нагрузочный имитатор обладает ограниченным набором функций, с его помощью невозможно настроить поведение виртуальных пользователей, а максимальное число запросов конечно, в связи с чем в Apache Bench не удается выполнить условие неограниченности времени экспериментов. Кроме того, это средство является устаревшим: в частности, оно поддерживает использование только одного потока.

2. Apache JMeter [10] — высокоуровневый нагрузочный имитатор с графическим интерфейсом. Это средство написано на Java и поддерживает большое число видов нагрузочного тестирования: объектами тестирования выступают не только HTTP-серверы, но и FTP-серверы, веб-сервисы, базы данных и т. д. Для нагрузочного тестирования HTTP-сервера можно задать такие параметры, как число пользователей, время достижения полной нагрузки, число итераций (в т. ч и бесконечно большое). В ходе проведения эксперимента в режиме реального времени отображаются значения различных метрик: число ошибок, скорость обработки

запросов, время ответа сервера и т. д. С помощью дополнительных параметров можно настроить сценарии поведения виртуальных пользователей: например, ввести случайную задержку между итерациями, то есть сымитировать ожидание на стороне пользователя.

Согласно официальной документации, после выполнения настройки Apache JMeter следует запускать с помощью консольного приложения. Это осуществляется с помощью следующей команды:

```
jmeter -n -t <файл с настройками тестирования>
```

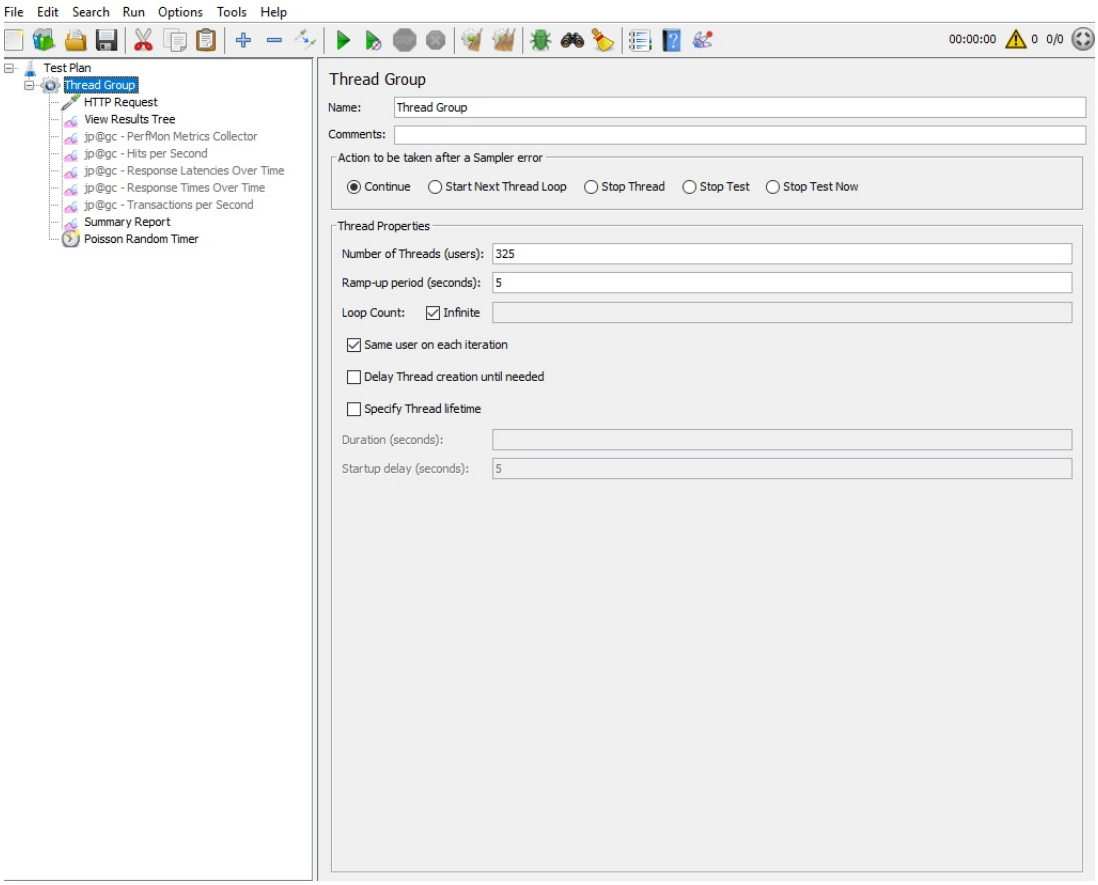

Рис. 2: Графический интерфейс программы Apache JMeter.

В целом Apache JMeter является универсальным средством для решения различных задач. Его недостатками являются сложность графического интерфейса и нестабильная работа при некоторых сочетаниях настроек.

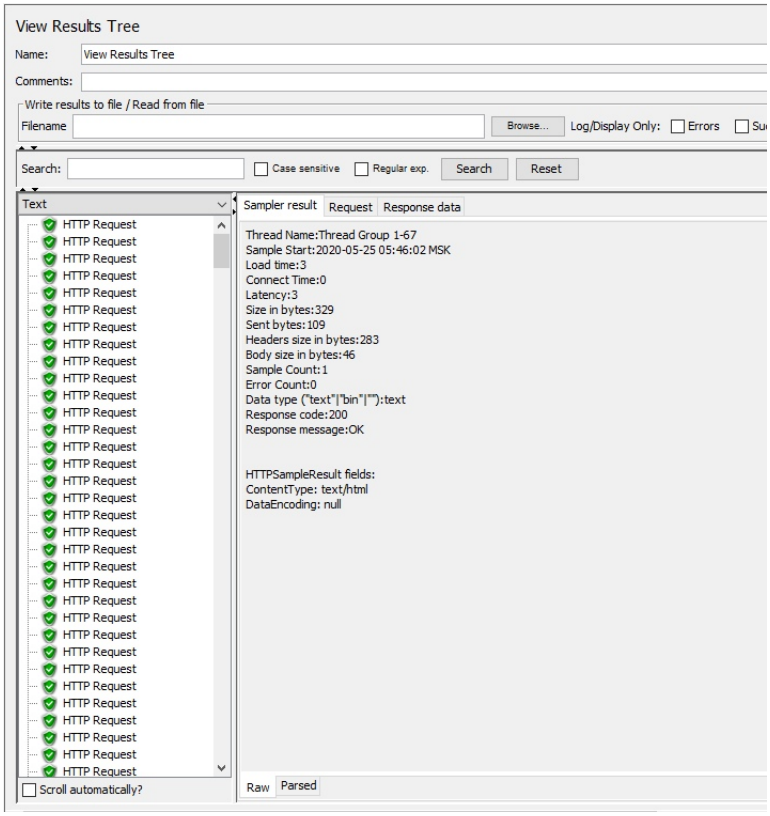

Рис. 3: Пример вывода работы программы Apache JMeter.

3. Locust [16] — современный консольный нагрузочный имитатор, реализованный в виде надстройки для языка Python [19]. Создание нагрузочных тестов производится посредством написания скрипта, который может быть написан с помощью любых средств языка. Таким образом, возможности данного нагрузочного имитатора ограничены только функциональностью и выразительностью языка Python. Скрипт запускается с помощью командной строки, и в веб-браузере по адресу `http://localhost:8080` становится доступным интерфейс запуска собственно теста. Оператору предлагается ввести общее число виртуальных пользователей, число новых пользователей в секунду и адрес веб-сервера для тестирования.

   Во время работы Locust в режиме реального времени собирает ряд метрик производительности сервера, которые выводятся на экран в том же веб-интерфейсе. Так, показывается скорость обработки запросов, процент успешных запросов, время ответа сервера. По умолчанию этот

нагрузочный имитатор работает до получения команды останова, что является его несомненным преимуществом в контексте данной работы. Однако в ходе работы с Locust не удалось добиться загрузки ЦП

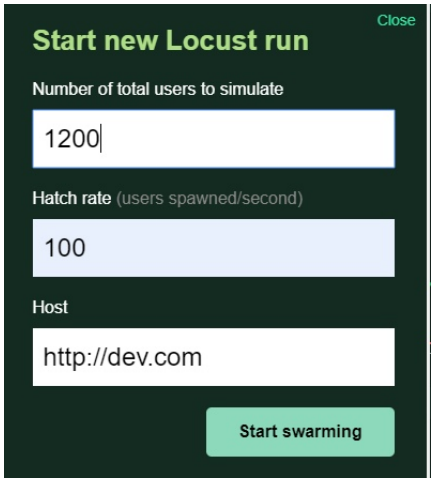

Рис. 4: Веб-интерфейс программы Locust.

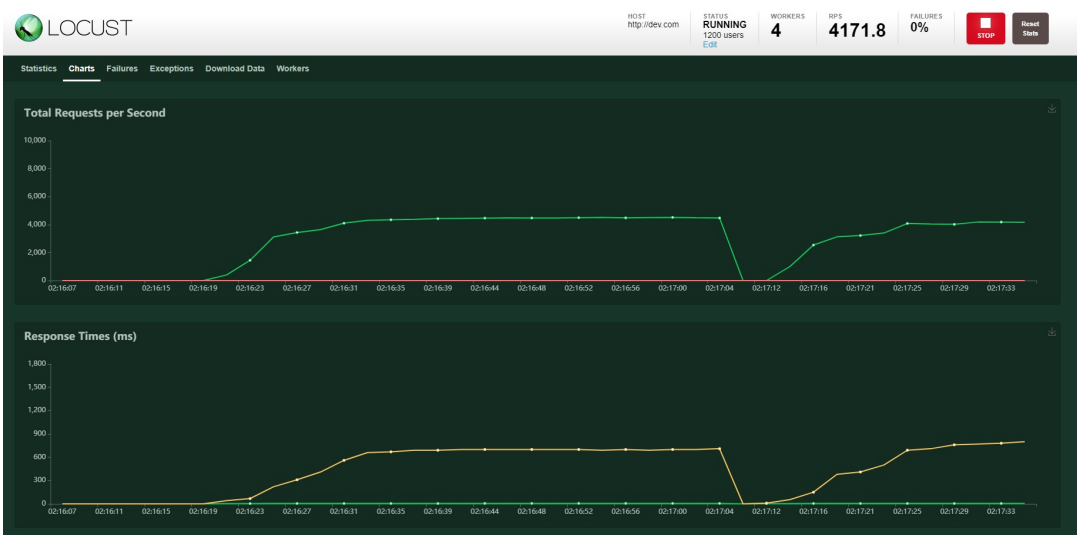

Рис. 5: Пример вывода работы программы Locust.

сервера более чем на 46%, даже несмотря на полное задействование вычислительных ресурсов устройства нагрузочного тестирования. Эту проблему можно решить, если использовать несколько устройств для нагрузочного тестирования, но в ограниченных условиях данной работы такой подход не был возможен.

Таким образом, выбор нагрузочного имитатора является нетривиальной задачей. С одной стороны, необходимо найти достаточно простой и универсальный инструмент, подходящий для тестирования экспериментального сервера; с другой стороны, необходимость проведения экспериментов с варьированием параметров накладывает на программные средства особые требования, удовлетворение которым подразумевает расширенный функционал.

Нагрузочный имитатор Locust, обладая большим количеством достоинств, в итоге оказался непригодным для проведения полных и состоятельных экспериментов. Поскольку ряд сценариев, доступных в Locust вследствие применения языка, можно реализовать и в Apache JMeter средствами графического интерфейса и при этом Apache JMeter позволяет достичь произвольной загрузки ЦП сервера вплоть до максимальной, для данного исследования было принято решение использовать нагрузочный имитатор Apache JMeter.

1.2 Подготовка среды нагрузочного тестирования

Ряд требований к организации нагрузочного тестирования, сформулированных выше, не представляется возможным удовлетворить с помощью выбора подходящего нагрузочного имитатора. В частности, представляет существенную проблему вопрос обеспечения невмешательства нагрузочного имитатора в работу веб-сервера. В ходе исследования было установлено, что при запуске нагрузочного имитатора и сервера на одном и том же устройстве каждая программа имеют свойство использовать как можно больше ресурсов, из-за чего обе программы становятся менее производительными. Кроме того, значения общей загрузки ЦП и использования ЦП процессами, связанными с веб-сервером, могут значительно отличаться: на практике общая загрузка ЦП может вдвое превышать загрузку ЦП, обеспечиваемую серверным ПО. Возникает ситуация, когда доступная мощность ЦП оказывается разделена между двумя программами, из-за чего эксперимент нельзя считать состоятельным, а условия — приближенными к реальным.

Чтобы преодолеть отмеченную трудность, необходимо разделить среды выполнения нагрузочного имитатора и веб-сервера. Этого можно добиться с использованием виртуальных машин, однако в рамках данной работы наи-

более логичным и простым решением оказалось использование двух вычислительных устройств: на одном запущено средство нагрузочного тестирования, а на другом — сервер. Устройство для работы нагрузочного имитатора представляет собой домашний компьютер с ОС Windows, мощность которого превосходит мощность ноутбука, используемого для работы сервера. Применение более мощного устройства для организации нагрузочного тестирования отвечает принятым практикам решения подобных задач.

Для организации взаимодействия двух компьютеров может использоваться как беспроводное, так и проводное соединение.

При беспроводном соединении компьютеры находятся в одной домашней Wi-Fi-сети. В этом случае недостатком является скорость этого взаимодействия, которая ограничена скоростью сети. Кроме того, работоспособность экспериментальной системы зависит от сторонних факторов — в частности, от сетевого маршрутизатора и его расположения. Из достоинств можно выделить отсутствие необходимости организации дополнительной сети.

При проводном соединении компьютеры соединяются с помощью Ethernet-кабеля, образуя локальную сеть. Для этого может использоваться сетевой коммутатор, однако в случае двух устройств его наличие необязательно. В таком варианте скорость передачи данных ограничена лишь стандартами используемых кабелей и сетевых карт. Недостатком является необходимость создания и установки локальной сети.

На практике оказалось, что при использовании беспроводной сети загрузка ЦП сервера в ходе эксперимента ограничена значением в 25%. При работе с локальной проводной сетью скорость обмена данных при 100%-ной загрузке ЦП значительно уступает максимально теоретической скорости, из чего сделать вывод, что сетевое оборудование никак не ограничивает как создание нагрузки, так и работу веб-сервера. Кроме того, в целях обеспечения бесперебойной работы нагрузочного имитатора и отсутствия дополнительных ограничений на взаимодействие нагрузочного имитатора и сервера антивирусное ПО и брандмауэр Windows должны быть отключены.

Таким образом, в данной работе используются два домашних компьютера, соединенные в локальную сеть посредством Ethernet-кабеля; нагрузочный имитатор запускается с наиболее мощного из компьютеров.

Глава 2. Проведение натурных экспериментов

В ходе подготовки к проведению экспериментов с выбранными ранее конкретными реализацией веб-сервера, нагрузочным имитатором, средой нагрузочного тестирования необходимо изучить устройство сервера, определить управляющие параметры, сформулировать дополнительные условия и настройки ПО.

2.1 Выбор конфигурационных параметров

Перечень доступных конфигурационных параметров для Apache HTTP Server зависит как от версии ПО, так и от используемой ОС и связан с программным устройством сервера.

Apache HTTP Server устроен как набор так называемых воркеров — неделимых рабочих объектов, каждый из которых обслуживает одного клиента. Каждый воркер может находиться в свободном или занятом состоянии [3]. Во время обработки запроса от клиента воркер поддерживает соединение с клиентом, находится в занятом состоянии, а после завершения обработки закрывает соединение и переходит в свободное состояние. Запросы от клиентов поступают в очередь на обслуживание, где ожидают, пока некоторый воркер не окажется в свободном состоянии.

Кроме того, современный HTTP-протокол поддерживает так называемые постоянные, или персистентные, соединения. Такие соединения не закрываются после обработки клиентского запроса. Вместо этого воркер переходит в особое состояние — состояние ожидания, в котором он недоступен для соединения с каким-либо другим клиентом, кроме того, чей запрос был недавно обработан. Технология, обеспечивающая данную возможность, носит название `KeepAlive`.

В силу особенностей ОС Windows Apache HTTP Server выполняется в двух процессах — основном и дочернем, и воркеры являются потоками дочернего процесса. Большинство параметров настраиваются в файлах `httpd-default.conf` и `httpd-mpm.conf`. Наиболее значимые параметры связаны именно с количеством воркеров и правилами работы постоянных соединений. Такие параметры по предположению могут существенно влиять на

работу сервера и, следовательно, рассматриваются в качестве потенциальных управляющих параметров.

В файле httpd-default.conf предлагается три настройки, определяющих работу персистентных соединений.

1. `KeepAlive` (KA) включает или выключает использование таких соединений. Доступными значениями являются `On` и `Off`.

2. `MaxKeepAliveRequests` ($MKAR$) задает максимальное число запросов от клиента в рамках одного персистентного соединения. После того как воркер обработал заданное число запросов, соединение принудительно закрывается, и воркер переходит в свободное состояние.

   Чем выше значение этого параметра, тем дольше каждый воркер будет обрабатывать только запросы от одного конкретного клиента, а запросы других клиентов будут перенаправляться другим воркерам. При наличии достаточно большого количества активных пользователей и меньшего числа воркеров запросы некоторых клиентов не будут обработаны. Естественно ожидать, что сервер будет обрабатывать меньше запросов в единицу времени и общая нагрузка на сервер снизится. Поэтому увеличение параметра MKAR должно приводить к уменьшению загрузки ЦП сервера.

   Доступные значения — целые числа от нуля. При нулевом значении персистентные соединения не закрываются независимо от числа запросов, что соответствует бесконечно большому значению $MKAR$. Из соображений соблюдения математической точности и логичности такое значение параметра не используется в экспериментах.

3. `KeepAliveTimeout` (KAT) задает максимальное время ожидания постоянного соединения в секундах и миллисекундах (при добавлении постфикса `ms`). Если соединение открыто, лимит в $MKAR$ запросов не был исчерпан, то оно принудительно закрывается после достижения указанного в настоящем параметре времени без новых запросов от клиента.

Чем больше это значение, тем дольше воркеры не обрабатывают запросы новых клиентов, а находятся в состоянии ожидания, что приводит к уменьшению нагрузки на сервер. Влияние этого параметра может быть различным в зависимости от параметра $MKAR$, т. к. при достаточно большой активности клиентов лимит в $MKAR$ запросов может быть достигнут задолго до истечения времени, заданного в KAT. Кроме того, в таком случае воркер не будет находиться в состоянии ожидания в течение достаточно долгого времени. Доступные значения — целые числа от одного в миллисекундах или секундах.

Файл httpd-mpm.conf отвечает за настройку модулей мультипроцессовой обработки (Multi-Processing Module, MPM [20]). Данные модули предназначены для организации одновременной обработки запросов пользователей. В каждый момент в Apache HTTP Server может быть включен только один модуль, и выбор модуля осуществляется как оператором, так и автоматически при установке ПО в зависимости от ОС. В ОС Windows единственным доступным модулем мультипроцессовой обработки является WinNT.

Для модуля WinNT в файле доступны две настройки:

1. `ThreadsPerChild` (TPC) задает число воркеров. Название объясняется вышеописанными особенностями устройства Apache HTTP Server при использовании ОС Windows, с учетом которого воркеры являются потоками дочернего процесса. Чем больше значение этого параметра, тем больше запросов могут быть обработаны в единицу времени, что должно увеличить загрузку ЦП в случаях, когда значение этого параметра меньше числа клиентов. Доступные значения — целые числа от 1 до 1920.

2. `MaxConnectionsPerChild` ($MCPC$) позволяет указать число открываемых воркерами соединений, после достижения которого происходит перезапуск дочернего процесса. Изменение этого параметра не должно оказывать влияния на загрузку ЦП. Во время перезапуска загрузка

ЦП будет уменьшаться вплоть до нуля, однако оно не входит во время работы сервера и, следовательно, загрузка ЦП в это время не должна измеряться.

Доступные значения — целые числа от нуля. При нулевом значении дочерний процесс не перезапускается, что соответствует бесконечно большому значению этого параметра. В ходе принятия решения об использовании этого параметра необходимо учесть этот факт и ограничить множество используемых значений, как и в случае с $MKAR$.

В ходе проведения экспериментов было исследовано, насколько сильно варьирование тех или иных параметров влияет на загрузку ЦП сервера.

Было установлено, что изменение параметра `KeepAliveTimeout` не оказывает заметного влияния на загрузку ЦП. Влияние наблюдается только при очень малых значениях, соответствующих времени ожидания персистентного соединения в несколько миллисекунд, в результате чего соединения закрываются настолько быстро, что запросы от клиентов сбрасываются. Загрузка ЦП при этом принимает значение, не зависящее от значения $MKAR$, что при разных $MKAR$ подразумевает как увеличение, так и уменьшение загрузки ЦП с увеличением значения параметра KAT. Таким образом, влияние данного параметра на загрузку ЦП является очень ограниченным и неоднозначным. Вероятно, такое наблюдение объясняется выбором сценариев экспериментов, при которых все виртуальные клиенты отправляют запросы с достаточно малыми интервалами.

Изменение параметра `ThreadsPerChild` показало выраженное влияние на загрузку ЦП в соответствии с предположением: пока значение TPC не достигло числа клиентов, а загрузка ЦП не достигла максимальной, увеличение значения на любое количество пунктов приводит к увеличению загрузки ЦП.

Варьирование параметра `MaxKeepAliveRequests` продемонстрировало, что он влияет на загрузку ЦП в достаточной мере, однако не так существенно, как TPC. Особенно существенное влияние наблюдается при малых значениях $MKAR$ — от 1 до 20: увеличение значения приводит к уменьшению загрузки ЦП. При других значениях параметра даже кратное увеличение значения не оказывает заметного влияния на загрузку ЦП.

Относительно параметра `MaxConnectionsPerChild` было установлено, что его изменение действительно не влияет на загрузку ЦП. Кроме того, как будет показано в следующем параграфе, при варьировании параметров оператору необходимо перезапускать дочерний процесс. Чтобы к запланированным перезапускам не добавлялись принудительные, значение этого параметра необходимо установить равным нулю.

Поскольку в рамках данной работы было решено выбрать два конфигурационных параметра, в качестве управлений были выбраны параметры TPC и $MKAR$. Для правильной работы параметра $MKAR$ установлено значение $KA =$ `On`. В дальнейшем из соображений наглядности управляющим переменным будут даны названия этих параметров: u_1 — TPC, u_2 — $MKAR$. Следует отметить, что параметр TPC приблизительно соответствует используемому в работе [3] параметру `MaxClients`.

2.2 Выбор режима работы сервера

Поскольку сервер, используемый в данной работе, является экспериментальным, его функциональность сводится к отрисовке статичной страницы. Таким образом, ответом на запросы пользователя является простой и детерминированный HTML-код.

Нагрузочный имитатор Apache JMeter позволяет запускать различные сценарии нагрузочного тестирования. В программе можно добавлять действия, выполняемые между итерациями запросов, в начале каждой итерации или в конце. По умолчанию предлагается настроить ряд параметров, которые можно видеть на рисунке 2: число виртуальных пользователей (`Number of Threads`, N), время достижения полной нагрузки (`Ramp-up period`, R). Включение настройки `Infinite` в поле `Loop Count` позволяет проводить тесты неограниченно долго.

Настройка `Same user on each iteration` определяет, создаются ли новые виртуальные пользователи на каждой итерации, то есть для каждого следующего запроса. Если эта настройка выключена, на каждой итерации пользователи новые, что приводит к тому, что персистентные соединения не используются. Хотя подобный сценарий в теории может подразумевать изме-

нение характера влияния параметров $MKAR$ и `KeepAliveTimeout`, в случае его применения нагрузочный имитатор не функционирует должным образом. Поэтому в условиях данной работы эта настройка остается включенной.

Поскольку для построения модели требуется постоянная нагрузка на сервер, время достижения полной нагрузки должно быть малым — например, 5 секунд. Особый интерес представляет параметр, отвечающий за число виртульных пользователей N.

При запуске нагрузочного тестирования с основными параметрами Apache JMeter виртуальные пользователи отправляют запросы на сервер так часто, как это возможно. Однако такой сценарий является достаточно грубым и не моделирует поведения реальных пользователей. Для обеспечения более реалистичного варианта нагрузочного тестирования запросы пользователей должны образовывать пуассоновский поток событий [2, 4]. С этой целью были применены дополнительные настройки Apache JMeter. Эти настройки позволяют добавить между итерациями случайную временную задержку, распределенную по экспоненциальному закону. Можно задать величину $^1/_\lambda$ — параметр распределения — и добавить постоянную задержку $dconst$.

Выбор конкретных значений этих настроек определяется рядом соображений. Стабильность работы Apache JMeter начинает нарушаться при увеличении N выше некоторого порогового уровня. Поскольку добавление задержки уменьшает создаваемую нагрузку, этот уровень сдвигается в сторону увеличения. Как было установлено, управляющий параметр TPC оказывает влияние на загрузку ЦП сервера до тех пор, пока его значение меньше N. Следовательно, при изменении задержки можно говорить о варьировании используемого множества значений TPC. При рассмотрении задачи о построении линейного приближения зависимости загрузки ЦП от управляющего параметра естественно добиваться того, чтобы множество значений варьируемого параметра имело максимально большую мощность. В противном случае дискретизация параметра окажет влияние на точность модели.

С другой стороны, существует предположение, что с увеличением N работа нагрузочного имитатора может стать менее стабильной даже с учетом пропорционального возрастания задержки между итерациями. Поэтому определение наиболее подходящих величин N, $dconst$ и $^1/_\lambda$ представляет из себя

поиск компромисса между точностью модели и стабильностью нагрузочного тестирования.

Кроме того, особое значение имеют факторы изменения влияния вариации управляющих параметров на загрузку ЦП. Эти факторы обеспечивают, в первую очередь, нелинейный характер этого влияния. Непосредственно для синтеза управления с обратной связью глубокое представление об этих факторах не является обязательным, потому что такое управление должно быть устойчиво к изменениям рабочей нагрузки; однако при исследовании возможности такого управления и сравнительной точности моделей на подобные явления следует обратить внимание.

Во-первых, при увеличении TPC в определенный момент влияние изменения $MKAR$ постепенно уменьшается, а при $TPC = N$ увеличение обоих параметров вообще не изменяет загрузки ЦП. Это можно объяснить наличием некоторой предельной нагрузки на сервер, определяемой настройками нагрузочного имитатора. В таком случае нагрузку уже нельзя увеличить посредством изменения параметров сервера.

Во-вторых, загрузка ЦП ограничена значением в 100%. Даже если предельная нагрузка на сервер еще не достигнута, при достижении загрузки ЦП в 100% изменение управляющих параметров не может привести к ее увеличению.

Данные факторы необходимо учитывать при определении подходящего режима нагрузочного тестирования для построения начальных условий эксперимента. Так, полезным представляется знать, до какого уровня можно увеличивать TPC, сохраняя характер влияния $MKAR$ на загрузку ЦП. В ходе экспериментов было установлено, что без упомянутых факторов увеличение $MKAR$ с 2 до 2000 приводит к уменьшению загрузки ЦП примерно на треть. Из предположения, что влияние TPC на загрузку ЦП носит характер, приближенный к линейному, первый фактор вступает в силу при TPC, равном $^2/_3$ от числа виртуальных пользователей.

В целях соблюдения достаточно малой нагрузки на устройство нагрузочного тестирования было выбрано $N = 325$. В этом случае изменение значения TPC от 1 до 200 сохраняет характер влияния параметра на загрузку ЦП. Таким образом, пороговым значением TPC является 200, а значени-

ем, по достижении которого влияние изменения параметров на загрузку ЦП нивелируется, — 325.

В ходе экспериментов было установлено, что при $MKAR = 1$ нагрузочный имитатор работает нестабильно. Поэтому данный параметр может варьироваться от 2 до 2000. Верхняя граница отрезка выбрана произвольно. Как было отмечено ранее, влияние изменения $MKAR$ на 1 на единицу загрузки ЦП зависит от значения $MKAR$, поэтому в экспериментах будет использоваться подмножество значений $MKAR$, включающее только сравнительно малые значения.

После решения вопроса о выборе множеств варьируемых значений управляющих параметров и N следует выбрать параметры задержки. Представляется полезным определить, при какой задержке предельная нагрузка на сервер совпадает с нагрузкой, обеспечивающей полную загрузку ЦП. В таком случае можно проводить эксперименты, в которых будет исключен и второй фактор изменения характера влияния управляющих параметров на загрузку ЦП. Эмпирическим путем установлено, что такие условия достигаются при ${}^1/_\lambda = 20$ мс, $dconst = 6$ мс и вышеуказанных значениях остальных параметров.

Таким образом, в качестве начальных условий эксперимента были выбраны следующие настройки:

$$N = 325, R = 5,$$
$$TPC \in [1, 200], MKAR \in [2, 2000],$$
$${}^1/_\lambda = 20, dconst = 6.$$

2.3 Ход эксперимента

Натурный эксперимент проводится следующим образом. После проверки готовности системы — состояния устройств, сети, отключения антивирусного ПО и брандмауэра — на более мощном компьютере запускается нагрузочный имитатор. Затем необходимое число раз повторяется следующий процесс:

1. Замеряется загрузка ЦП.

2. Изменяются значения управляющих параметров.

3. Проверяется готовность сервера.

Каждая итерация процесса соответствует одному шагу дискретного времени в уравнении (1).

Замеры значений загрузки ЦП можно производить разными способами. При неизменных значениях управляющих параметров загрузка ЦП немного колеблется в силу возмущений, вызванных работой ОС, особенностей устройства сервера. Кроме того, сам процесс измерения нагрузки ЦП может быть реализован по-разному и влиять на получаемые значения, хотя это влияние представляется незначительным. Поэтому для того чтобы результаты эксперимента были состоятельными, следует в рамках каждой итерации последовательно измерять значения загрузки ЦП через равные промежутки времени, а затем на основе полученных значений вычислять загрузку ЦП на данной итерации, то есть $y(k)$ [2]. Выбор способа вычисления $y(k)$ может осуществляться различным образом, однако особый интерес представляют подсчет арифметического среднего и медианы среди полученных результатов.

Арифметическое среднее является наиболее естественным способом сглаживания выходного сигнала; эта функция позволяет нивелировать небольшие флуктуации загрузки ЦП. Вычисление медианного значения представляется полезным в случаях, когда загрузка ЦП может резко изменяться в течение итерации или зависит от времени, прошедшего с момента начала итерации. Несмотря на преимущества подсчета медианы значений загрузки ЦП, такой способ может не давать желаемого результата в случаях, когда загрузка ЦП невелика и среди измеренных значений много нулевых. Поэтому выбора способа вычисления $y(k)$ по измеренным значениям зависит еще и от программной реализации экспериментальной системы. Использование других функций, таких как взвешенное среднее, не рассматривается в данной работе.

Варьирование управляющих параметров должно производиться так, чтобы каждый из параметров принимал как можно больше значений из своего

множества значений. Поскольку целью эксперимента является исследование работы сервера в динамике, а предполагаемая модель сервера имеет вид (1), параметры нужно изменять как в сторону увеличения, так и в сторону уменьшения. В соответствии с идеей, сформулированной в [3], в данной работе вариация параметров производится по траектории синусоиды. То есть, если множество значений параметра u_i представляет собой отрезок $[a, b]$, то имеет место формула:

$$u_i(k) = \left\lfloor \frac{(b+a)}{2} - \frac{b-a}{2} \cos\left(\frac{2\pi k}{n}\right) \right\rfloor, \qquad k = 1,\ 2,\ ...,\ n, \qquad (2)$$

где n — число шагов синусоиды. Таким образом, на каждой итерации управляющие параметры изменяются в конфигурационных файлах Apache HTTP Server в соответствии с формулой (2). При одновременном варьировании двух параметров периоды соответствующих им синусоид должны соотноситься как взаимно простые числа, чтобы обеспечить равномерное покрытие множеств значений управляющих параметров.

После изменения значений параметров необходимо осуществить перезапуск дочернего процесса сервера. Во время перезапуска уже установленные соединения не закрываются, но новые запросы не обрабатываются. Перезапуск занимает чуть более секунды. Это время следует исключить из периода замера загрузки ЦП. Перезапуск осуществляется исполнением в командной строке следующей команды:

```
httpd -k restart
```

Несмотря на то что перезапуск в любом случае происходит при исполнении команды, новые параметры применяются только при работе с командной строкой в режиме администратора, что необходимо учесть при проведении эксперимента и программной реализации.

В силу необходимости перезапуска дочернего процесса сервера можно говорить об ограничениях частоты действий управления. Такие ограничения, в частности, обусловливают применение методов сглаживания выходного сигнала, а не разовое его измерение.

2.4 Построение модели

Результатом проведения конкретного эксперимента являются три набора данных: $\{TPC(1), TPC(2), ..., TPC(r)\}$,
$\{MKAR(1), MKAR(2), ..., MKAR(r)\}$,
$\{\hat{y}(1), \hat{y}(2), ..., \hat{y}(r), \hat{y}(r+1)\}$. На основе этих данных можно выбрать рабочую точку, которая позволит построить достаточно точное линейное приближение модели веб-сервера в форме (1). Одним из подходов является выбор средних значений входных и выходного сигналов в качестве рабочей точки, то есть:

$$\overline{y} = \frac{1}{r}\sum_{k=2}^{r+1}\hat{y}(k), \tag{3}$$

$$\overline{TPC} = \frac{1}{r}\sum_{k=1}^{r}TPC(k), \quad \overline{MKAR} = \frac{1}{r}\sum_{k=1}^{r}MKAR(k).$$

Вектор $\boldsymbol{h}$ в случае конкретного эксперимента определен и имеет постоянное значение при любом k, поэтому в дальнейших выкладках опускается. Следует отметить, что пределы суммирования для выходного сигнала отличаются от таковых для входных сигналов, поскольку выходной сигнал сравнивается с результатом вычисления модели системы.

Пусть $\widetilde{TPC}(k) \triangleq TPC(k) - \overline{TPC}$, $\widetilde{MKAR}(k) \triangleq MKAR(k) - \overline{MKAR}$, $\tilde{y}(k) \triangleq \hat{y}(k) - \overline{y}$. Необходимо выбрать коэффициенты уравнения (1): α_1, β_1, β_2. Это можно осуществить с помощью метода наименьших квадратов [3]. С этой целью следует найти минимум функции:

$$J(\alpha_1, \beta_1, \beta_2) = \sum_{k=1}^{r}\left[e\left(k+1\right)\right]^2,$$

где $e\left(k+1\right) \triangleq \tilde{y}\left(k+1\right) - \alpha_1\tilde{y}\left(k\right) - \beta_1\widetilde{TPC}\left(k\right) - \beta_2\widetilde{MKAR}\left(k\right)$ В терминах регрессионного анализа решением данной задачи является матричное выражение

$$\boldsymbol{b} = \left(\boldsymbol{X}^{\intercal}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\intercal}\boldsymbol{y}, \tag{4}$$

где

$$\boldsymbol{b} = (\alpha_1, \beta_1, \beta_2)^\intercal, \quad \boldsymbol{X} = \begin{pmatrix} \widetilde{TPC}(1) & \widetilde{MKAR}(1) & \tilde{y}(1) \\ \widetilde{TPC}(2) & \widetilde{MKAR}(2) & \tilde{y}(2) \\ \cdots & \cdots & \cdots \\ \widetilde{TPC}(r) & \widetilde{MKAR}(r) & \tilde{y}(r) \end{pmatrix},$$

$$\boldsymbol{y} = (\tilde{y}(2), \tilde{y}(3), ..., \tilde{y}(r+1))^\intercal.$$

После получения коэффициентов $\boldsymbol{b}$ возникает вопрос оценивания точности модели. В данной работе, в соответствии с указанными в [3] соображениями, используются две метрики: среднеквадратическая ошибка $RMSE$ и коэффициент детерминации R^2. Вычисление их осуществляется по указанным ниже формулам:

$$RMSE = \sqrt{\frac{1}{r}\sum_{k=1}^{r}\left[e(k+1)\right]^2} \tag{5}$$

$$R^2 = 1 - \frac{D(e)}{D(\tilde{y})}, \tag{6}$$

где D означает дисперсию набора значений.

Таким образом, процесс построения модели представляет собой последовательное решение нескольких числовых задач. Все вышеописанные вычисления легко автоматизировать с помощью программных средств.

Глава 3. Программная реализация системы

В настоящей главе описывается программный продукт, который был создан с целью автоматизации проведения натурных экспериментов. Исходный код программы был написан на языке Python 3 с использованием различных библиотек, в том числе Numpy [7], позволяющей упростить выполнение матричных расчетов. Фрагменты кода представлены в Приложении А.

Необходимые средства работы с Apache HTTP Server реализованы в виде метода класса `Server`. Так, обращаясь к этим методам, можно запустить, остановить или перезапустить сервер (методы `start`, `stop`, `restart` соответственно). В ходе реализации команды автоматически исполнялись в командной строке с предварительной проверкой наличия повышенных привилегий пользователя. Отдельные методы служат для работы с конфигурационными файлами Apache HTTP Server.

Несмотря на наличие в открытом доступе ряда библиотек, предназначенных непосредственно для изучения (парсинга) конфигурационных файлов сервера, в ходе работы над программой наиболее удобным решением оказалось использование регулярных выражений, которые позволяют обращаться к конкретным частям текста в файлах, соответствующим числовым значениям управляющих параметров. В терминах языка Python использованные регулярные выражения имеют вид `?<=>\n\tThreadsPerChild\s)\s*\d{1,4}` и `(?<=MaxKeepAliveRequests\s)\s*\d{1,4}` для параметров TPC и $MKAR$ соответственно. Следует отметить, однако, что подобный подход не является масштабируемым.

Измерение выходного сигнала — загрузки ЦП — осуществляется с помощью библиотеки psutils [8]. Существует два подхода к измерению загрузки ЦП: измерение использования ЦП процессом сервера и общей загрузки ЦП. Измерение общей загрузки ЦП в большей степени соответствует практикам построения управления в контуре обратной связи, поскольку не требует знания внутреннего устройства Apache HTTP Server, а внешние возмущения могут выражаться в работе ОС или фоновых программ.

Однако с осуществлением такого измерения возникли трудности: библиотека psutils реализована таким образом, что с ее помощью не удается

получить данные об общей загрузке ЦП. Выводимые значения оказываются заниженными. В связи с этим было принято решение в рамках данной работы измерять потребление ресурсов ЦП дочерним процессом сервера. При соблюдении сформулированных в параграфе 1.2 условий нагрузочного тестирования фактические значения двух метрик не должны существенно отличаться. Кроме того, возмущения в виде работы фоновых программ в теории могут повлиять и на потребление ресурсов ЦП процессами сервера. В ходе проведения экспериментов отслеживалось, чтобы на компьютере работало как можно меньше фоновых процессов, было отключено беспроводное соединение, а сама программа запускалась так, чтобы оказывать наименьшее возможное влияние на загрузку ЦП.

Дочерний процесс Apache HTTP Server прекращает работу во время каждого перезапуска сервера и создается заново, при этом его идентификатор (PID) меняется. Следовательно, в программе должно быть реализовано определение актуального PID дочернего процесса сервера. В данной программе эта переменная представляется как поле класса `Server`, его значение перезаписывается после каждого перезапуска сервера.

Вычисление агрегированной загрузки ЦП на каждой итерации, то есть $\hat{y}(k)$, реализовано как метод класса `Server` (`get_cpu_usage`). Посредством задания входных аргументов можно определить, с каким интервалом будут производиться замеры и какая функция будет использована для сглаживания значений: арифметическое среднее или медиана. Из-за несовершенства библиотеки psutils при измерении в редких случаях может быть получено значение больше 100%, которое не входит в множество значений загрузки ЦП. Поэтому в таких случаях значение функции корректируется до 100%.

Непосредственно функционал программы сводится к двум основным возможностям: автоматическое проведение экспериментов и автоматическое построение модели по полученным данным.

Автоматическое проведение экспериментов с возможностью вывода графиков в реальном времени реализовано в методе класса `Server` (`vary_config`). В аргументах функции задается число управляющих параметров, границы множества значений каждого из них, число шагов синусоидальной волны (2) и число таких волн. Если выбран только один из параметров

TPC и $MKAR$, требуется указать значение по умолчанию для второго. Для одного из параметров число волн может быть вычислено автоматически, при этом соблюдается условие взаимной простоты количеств волн (и, следовательно, их периодов). Генерация самих синусоид производится во вспомогательной функции `sine_wave`. В целях обеспечения равномерного покрытия множества значений параметров (полуинтервалов ниже и выше значения, входящего в рабочую точку) каждая синусоида генерируется в два этапа: сначала вычисляется $\frac{n}{2}$ значений на отрезке $[0, \pi]$, затем — $\frac{n}{2}$ значений на отрезке $[\pi, 2\pi]$.

Кроме того, задаются длительность каждой итерации, интервал замеров загрузки ЦП и настройки вывода графиков. Предоставляется возможность добавить время простаивания после перезапуска сервера, в течение которого процесс перезапуска считается незавершенным и замеры загрузки ЦП не производятся. Число итераций r определяется равным произведению числа шагов синусоиды на число синусоидальных волн. На начальных этапах выполнения функции выполняются проверка входных аргументов, вычисление числа волн при необходимости, вывод подсказок на экран. Затем осуществляются установка начальных управляющих параметров и перезапуск сервера. В основном цикле функции производится непосредственно вычисление данных, которое можно разбить на следующие шаги:

1. значение $\hat{y}(k)$ вычисляется и сохраняется в структуре данных;
2. определяются значения $TPC(k)$, $MKAR(k)$ в соответствии со сгенерированными синусоидальными волнами;
3. при необходимости производится перезапуск дочернего процесса сервера;
4. при необходимости обновляются графики.

На третьем шаге перезапуск производится только в том случае, если значения $TPC(k) \neq TPC(k-1)$ или $MKAR(k) \neq MKAR(k-1)$ После завершения всех итераций $r+1$-мерные векторы $\hat{\boldsymbol{y}}, \boldsymbol{TPC}, \boldsymbol{MKAR}$ сохраняются в форме структур данных — массивов Numpy. На основе этих массивов формируется

и сохраняется файл в формате CSV, в котором по столбцам записываются значения k, $TPC(k)$, $MKAR(k)$, $\hat{y}(k)$.

Построение модели по данным из CSV-файла реализовано посредством ряда функций. Функция `read_csv` позволяет сохранить данные эксперимента в массивах Numpy с использованием всех или некоторых управляющих параметров. Функция `plot_csv` реализует построение графика по данным эксперимента. Определяется рабочая точка $(\overline{y}, \overline{\boldsymbol{u}})$ по формуле (3). В функции `least_squares_arma` производится вычисление коэффициентов α_1, β_1, β_2 посредством встроенного в библиотеку Numpy метода в соответствии с формулой (4) метода наименьших квадратов. По вычисленным коэффициентам и данным эксперимента в функции `fit` вычисляются аналитические значения:

$$\tilde{y}^*(k+1) \triangleq \alpha_1 \tilde{y}(k) + \beta_1 \widetilde{TPC}(k) + \beta_2 \widetilde{MKAR}(k), \quad k = 1, 2, \ldots r$$

В этой же функции осуществляется построение диаграммы рассеяния: по горизонтальной оси откладываются значения $\tilde{y}(k+1)$, по вертикальной — значения $\tilde{y}^*(k+1)$. Диаграмма сравнивается с линией $\tilde{y}(k+1) \equiv \tilde{y}^*(k+1)$: чем диаграмма рассеяния больше похожа на эту линию, тем точнее модель описывает динамику загрузки ЦП.

Функции `rmse` и `r_squared` позволяют найти значения соответственно метрик $RMSE$ (5) и R^2 (6). С помощью функции `print_equation` уравнения динамики сервера выводятся на экран в форме (1).

Программу можно запускать, как вручную выполняя код, так и в варианте консольного приложения. В последнем случае доступен ряд аргументов командной строки, в том числе регулирующих путь к генерируемому CSV-файлу, вывод графиков и отладочной информации, запрос о подтверждении начала эксперимента. Настройки эксперимента при таком запуске передаются в формате JSON, путь к которому также указывается в аргументах командной строки. Перед запуском программы необходимо убедиться, что соблюдаются все условия эксперимента, а нагрузочный имитатор запущен и генерирует нагрузку с желаемыми параметрами.

Глава 4. Результаты моделирования

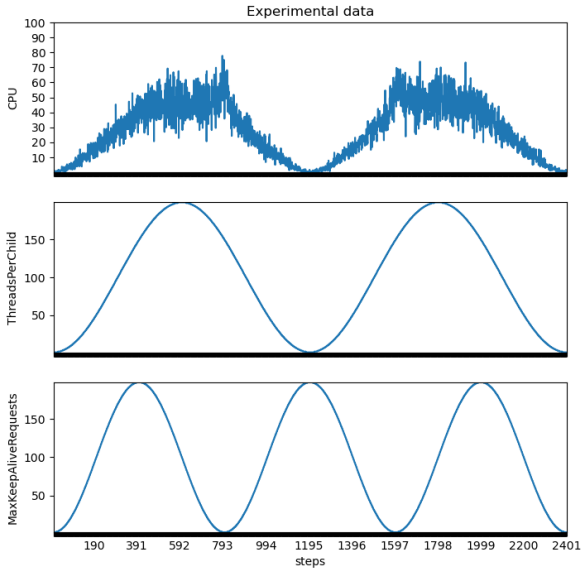

Рис. 6: Ход эксперимента 1.

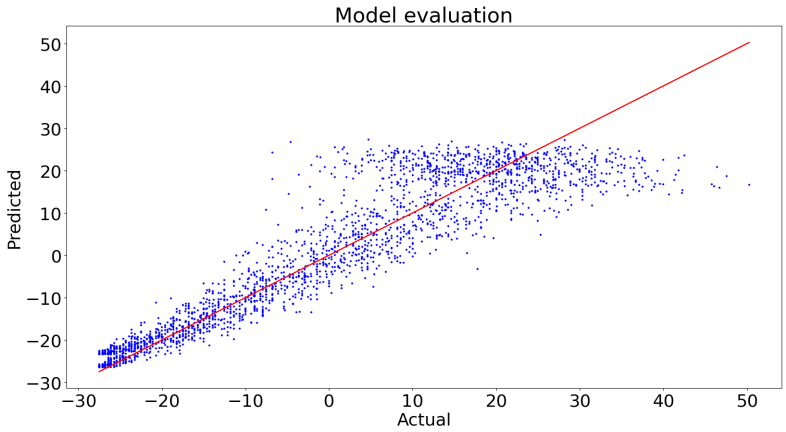

Рис. 7: Диаграмма точности модели, построенной в результате эксперимента 1.

В рамках данной работы проводились натурные эксперименты. Программа, описанная в главе 3, запускалась с различными параметрами. Запуск программы производился в варианте консольного приложения без отрисовки графиков в режиме реального времени с целью минимизировать влияние ра-

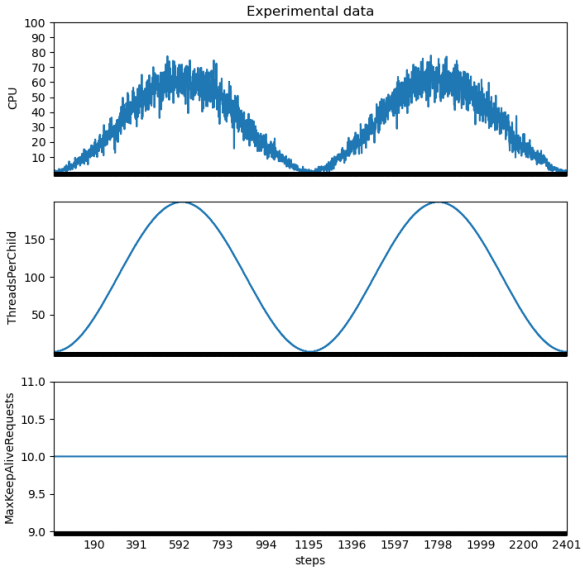

Рис. 8: Ход эксперимента 2.

боты программы на измеряемые значения загрузки ЦП. Ниже представлены результаты проведенных экспериментов.

Каждому эксперименту соответствует несколько графиков. Сначала выводятся три графика, содержащие информацию о ходе эксперимента. На верхнем графике представлено изменение загрузки ЦП в зависимости от итерации, на среднем и нижнем графиках — изменение TPC и $MKAR$ в зависимости от итерации. Средний и нижний графики, таким образом, представляют собой несколько синусоидальных волн, если производится вариация обоих параметров, и синусоидальные волны и прямую, если варьируется один из параметров. В ходе построения модели выводится диаграмма рассеяния, позволяющая сформировать мнение о точности модели.

1. $TPC \in [1, 199], MKAR \in [2, 198], r = 2400$; число волн для TPC — 2, для $MKAR$ — 3, вычисление $\hat{y}(k)$ производится посредством усреднения.

   Для полученных данных $\overline{y} = 27{,}535, \overline{TPC} = 100, \overline{MKAR} = 100$;

   Построенное уравнение динамики сервера имеет вид $\tilde{y}^*(k+1) = 0{,}168\,\tilde{y}(k) + 0{,}204\,\widetilde{TPC}(k) - 0{,}016\,\widetilde{MKAR}(k)$;

   Метрики точности модели: $R^2 = 0{,}86, RMSE = 6{,}99$;

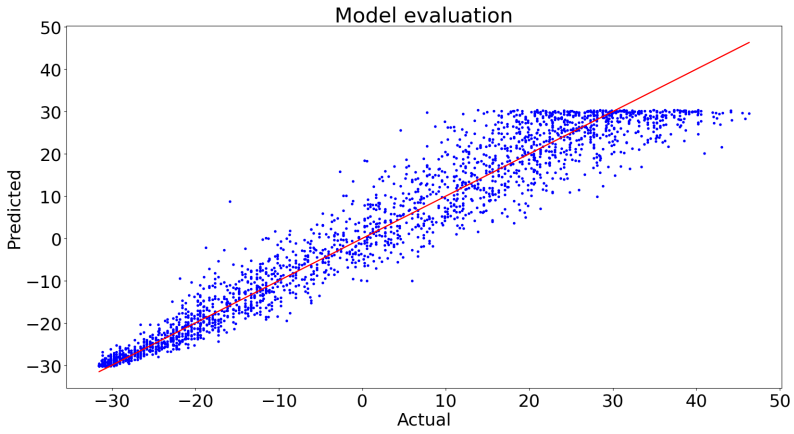

Рис. 9: Диаграмма точности модели, построенной в результате эксперимента 2.

Графики представлены на рис. 6, 7.

Если при построении модели не учитывать вариацию $MKAR$, то уравнение примет вид

$$\tilde{y}^*(k+1) = 0{,}195\,\tilde{y}(k) + 0{,}198\,\widetilde{TPC}.$$

Такая модель имеет лишь немногим меньшую точность:

$$R^2 = 0{,}856, RMSE = 7{,}08$$

В этом случае можно составить уравнение для пропорционального регулятора сервера, исходя из соотношения $\tilde{y}^*(k+1) = 0$:

$$0{,}198\,\widetilde{TPC}(k) = -0{,}195\,\tilde{y}(k),$$

откуда

$$TPC(k) = -\frac{0{,}195}{0{,}198}\,(y\,(k) - \overline{y}) + \overline{TPC}(k),$$

что приблизительно соответствует

$$TPC(k) = \overline{TPC}(k) + \overline{y} - y\,(k)\,.$$

2. $TPC \in [1, 199], MKAR \equiv 10, r = 2400$; число волн для TPC — 2,

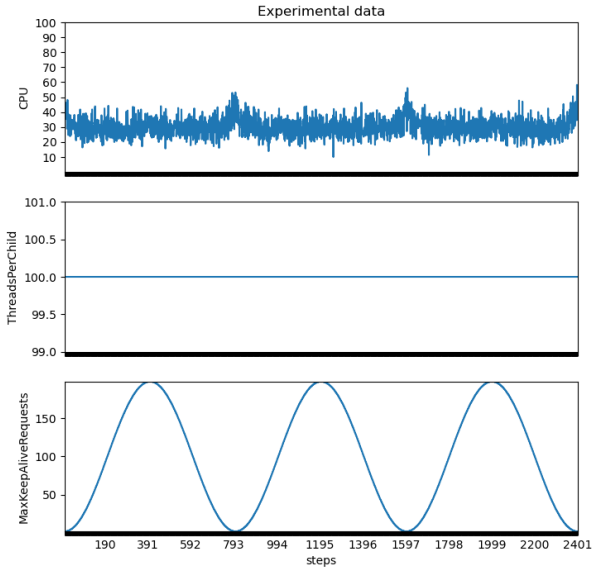

Рис. 10: Ход эксперимента 3.

вычисление $\hat{y}(k)$ производится посредством усреднения.
Для полученных данных $\overline{y} = 31{,}554, \overline{TPC} = 100$;
Построенная модель имеет вид $\tilde{y}^*(k+1) = 0{,}017\,\tilde{y}(k) + 0{,}3\,\widetilde{TPC}$;
Метрики точности модели: $R^2 = 0{,}94, RMSE = 5{,}26$;
Графики представлены на рис. 8, 9.

В данном эксперименте производилась вариация только параметра TPC. Построенная модель имеет достаточно высокую точность — выше, чем при использовании модели с одним управляющим параметром для данных предыдущего эксперимента. Уравнение динамики сервера указывает на то, что изменение TPC влияет на загрузку ЦП в большой степени. Уравнение для пропорционального регулятора имеет вид:

$$TPC(k) = \overline{TPC}(k) - 0{,}57\,(y\,(k) - \overline{y})\,.$$

3. $TPC \equiv 100, MKAR \in [2, 198], r = 2400$; число волн для $MKAR$ — 3, вычисление $\hat{y}(k)$ производится посредством усреднения.
Для полученных данных $\overline{y} = 30{,}012, \overline{MKAR} = 100$;
Построенная модель имеет вид $\tilde{y}^*(k+1) = 0{,}140\,\tilde{y}(k) - 0{,}014\,\widetilde{MKAR}$;

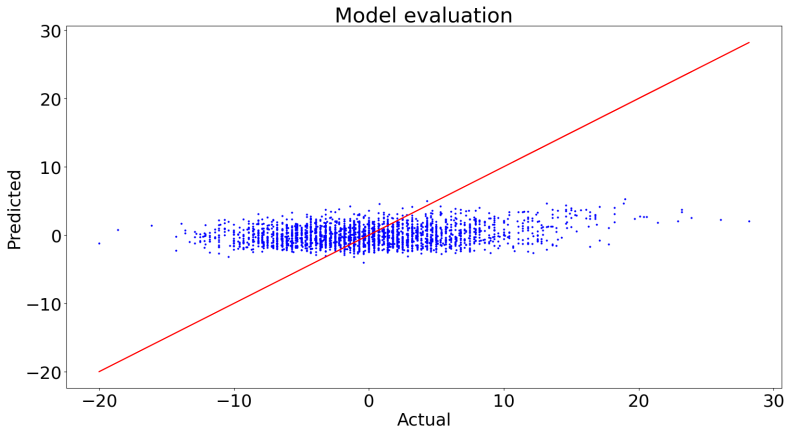

Рис. 11: Диаграмма точности модели, построенной в результате эксперимента 3.

Метрики точности модели: $R^2 = 0{,}05$, $RMSE = 5{,}85$;
Графики представлены на рис. 10, 11.

В данном эксперименте производилась вариация только параметра $MKAR$. Получившееся уравнение показывает чрезвычайно слабую зависимость загрузки ЦП от управляющего параметра в окрестности рабочей точки; точность модели также очень низка. Это наблюдение указывает на непригодность использования $MKAR$ в качестве единственного управляющего параметра при определенных в данной работе условиях экспериментов. При сужении множества варьируемых значений $MKAR$ до $[2, 16]$ уравнение приобретает вид:

$$\tilde{y}^*(k+1) = 0{,}091\,\tilde{y}(k) - 0{,}546\,\widetilde{MKAR}.$$

Изменения в уравнении характеризуют многократное увеличение влияния варьирования $MKAR$ на загрузку ЦП (α_1), однако значения метрик точности остаются неудовлетворительными:

$$R^2 = 0{,}2,\ RMSE = 6{,}14.$$

4. $TPC \in [151, 499]$, $MKAR \in [2, 198]$, $r = 2400$; число волн для TPC — 2, для $MKAR$ — 3, вычисление $\hat{y}(k)$ производится посредством усреднения.

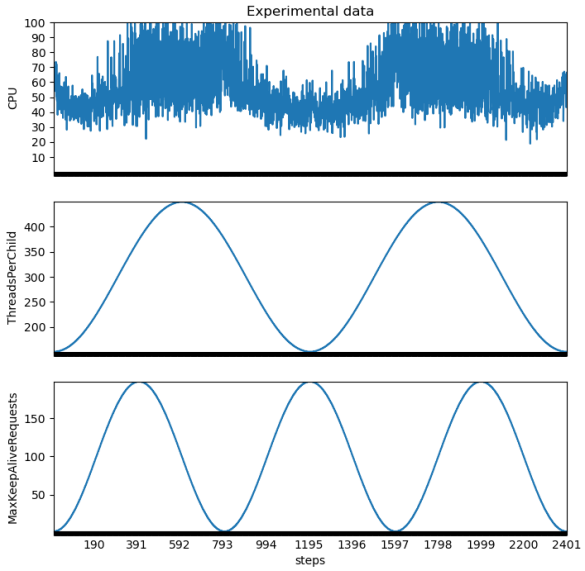

Рис. 12: Ход эксперимента 4.

Для полученных данных $\overline{y} = 55{,}081, \overline{TPC} = 300, \overline{MKAR} = 100$;
Построенное уравнение динамики сервера имеет вид $\tilde{y}^*(k+1) = -0{,}157\,\tilde{y}(k) + 0{,}096\,\widetilde{TPC}(k) - 0{,}033\,\widetilde{MKAR}(k)$;
Метрики точности модели: $R^2 = 0{,}27, RMSE = 15{,}13$;
Графики представлены на рис. 12, 13.

В ходе этого эксперимента была исследована динамика сервера при значениях управляющего параметра TPC, соответствующих высокой загрузке ЦП. Как было показано выше, при настоящих условиях эксперимента возникает нелинейность, обусловленная сразу несколькими факторами. Так, при $TPC > 200$ влияние изменения $MKAR$ на загрузку ЦП ослабевает; при $TPC > 325$ достигается предельный уровень нагрузки, и влияние управляющих параметров уменьшается до нуля. Кроме того, при $TPC > 200$ нелинейность возникает из-за достижения загрузки ЦП в 100%. Вследствие этих факторов построенная по результатам эксперимента модель оказалась недостаточно точной. Уменьшение влияния вариации параметров отражается в сравнительно малых значениях коэффициентов β_1, β_2, а наличие ограничения загрузки ЦП — в отрицательном коэффициенте α_1.

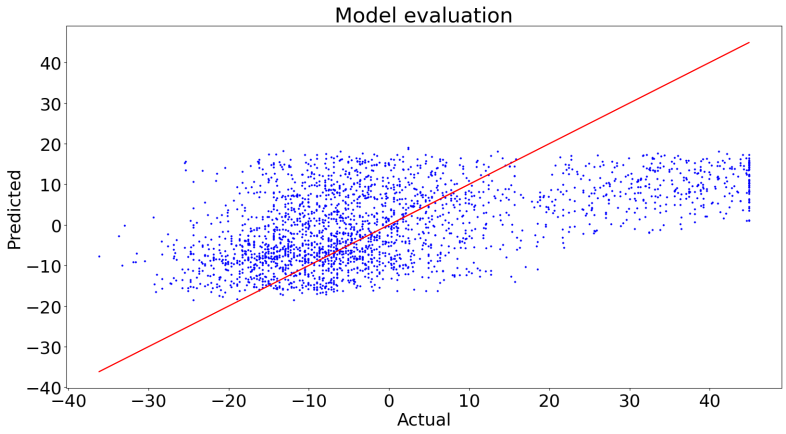

Рис. 13: Диаграмма точности модели, построенной в результате эксперимента 4.

Следует отметить, что подобное сочетание нелинейностей является искусственно созданным примером, для которого построение линейного приближения особенно затруднительно. Однако точность модели можно увеличить, произведя сужение множества значений управляющих параметров. В дальнейших экспериментах в качестве управляющего параметра используется только TPC.

5. $TPC \in [1, 499], MKAR \equiv 10, r = 3000$; число волн для TPC — 2, вычисление $\hat{y}(k)$ производится посредством усреднения.
   Для полученных данных $\overline{y} = 46{,}756, \overline{TPC} = 250$;
   Построенное уравнение динамики сервера имеет вид $\tilde{y}^*(k+1) = -0{,}019\,\tilde{y}(k) + 0{,}153\,\widetilde{TPC}(k)$;
   Метрики точности модели: $R^2 = 0{,}81, RMSE = 12{,}9$;
   Графики представлены на рис. 14, 15.

   В данном эксперименте TPC варьировался в самом широком диапазоне, включающем как линейные, так и нелинейные участки. Несмотря на потенциальные сложности построения линейного приближения для множеств варьируемых значений такой мощности, точность модели оказалась достаточно большой, а коэффициенты вполне адекватно отражают динамику сервера. Одним из признаков наличия нелинейных участков множеств варьируемых значений является гетероскедастичность — неравномерный характер диаграммы рассеяния [3].

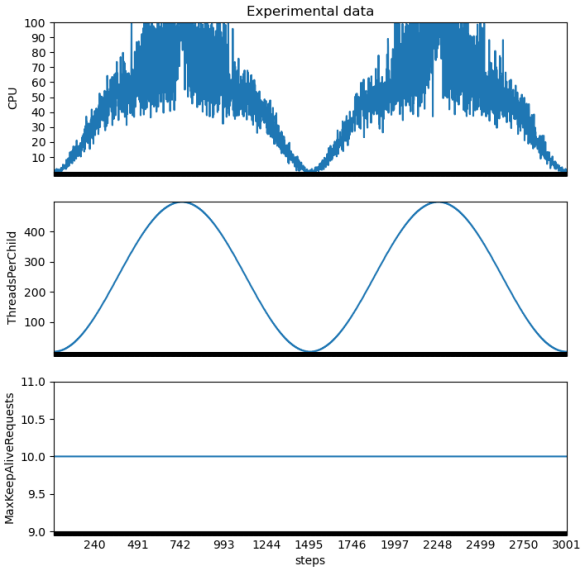

Рис. 14: Ход эксперимента 5.

6. ${}^{1}/_{\lambda} = 40$ мс, $dconst = 12$ мс;
   $TPC \in [1, 399], MKAR \equiv 10, r = 36$; число волн для TPC — 2, вычисление $\hat{y}(k)$ производится посредством усреднения.
   Для полученных данных $\overline{y} = 28{,}603, \overline{TPC} = 200$;
   Построенное уравнение динамики сервера имеет вид $\tilde{y}^*(k+1) = 0{,}158\,\tilde{y}(k) + 0{,}108\,\widetilde{TPC}(k)$;
   Метрики точности модели: $R^2 = 0{,}92, RMSE = 5{,}44$;
   Графики представлены на рис. 16, 17.

   В ходе этого эксперимента были использованы измененные настройки нагрузочного имитатора: параметры случайной задержки между запросами виртуальных пользователей были скорректированы таким образом, чтобы нагрузка на сервер снизилась примерно вдвое. Использовалось малое число значений TPC, взятых из широкого диапазона. Исходя из полученного уравнения, можно видеть, что влияние изменения TPC на загрузку ЦП уменьшилось, а $\alpha_1 > \beta_1$, что может указывать как на возникновение дополнительных причин нелинейности динамики сервера, так и на недостаточное покрытие множества варьируемых значений.

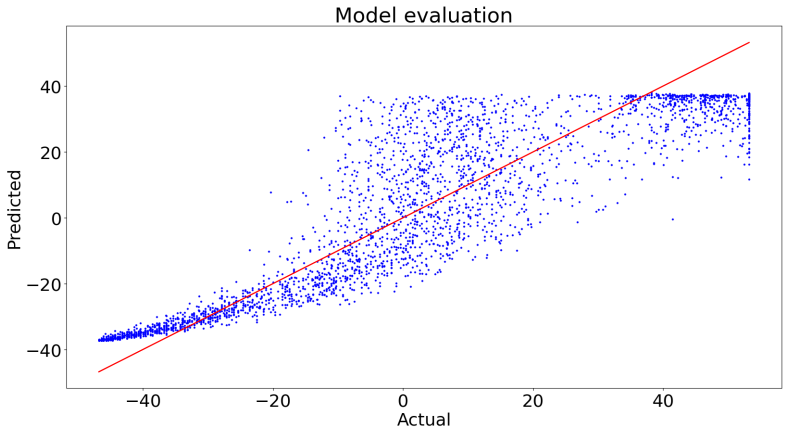

Рис. 15: Диаграмма точности модели, построенной в результате эксперимента 5.

7. ${}^{1}/_{\lambda} = 10$ мс, $dconst = 3$ мс;
   $TPC \in [1, 199], MKAR \equiv 10, r = 72$; число волн для TPC — 2, вычисление $\hat{y}(k)$ производится посредством усреднения.
   Для полученных данных $\overline{y} = 53{,}440, \overline{TPC} = 100$;
   Построенное уравнение динамики сервера имеет вид $\tilde{y}^*(k+1) = -0{,}113\,\tilde{y}(k) + 0{,}558\,\widetilde{TPC}(k)$;
   Метрики точности модели: $R^2 = 0{,}97, RMSE = 5{,}83$;
   Графики представлены на рис. 18, 19.

   В ходе этого эксперимента случайная задержка между запросами виртуальных пользователей была уменьшена, нагрузка на сервер повысилась примерно вдвое. Из полученного уравнения следует, что влияние изменения TPC на загрузку ЦП значительно увеличилось, а $\alpha_1 < 0$, как и для модели, полученной в эксперименте 5. Коэффициенты уравнения представляются как адекватно отражающие характер динамики сервера, несмотря на использование небольшого количества итераций.

На основании данных, полученных при проведенном исследовании, можно сделать ряд наблюдений. Во-первых, на базе уравнения динамики сервера можно построить уравнение пропорционального регулятора для непосредственного управления системой с помощью обратной связи. Во-вторых, параметр $MKAR$ плохо подходит для организации такого управления. В-третьих, наличие факторов нелинейности тем не менее не исключает возмож-

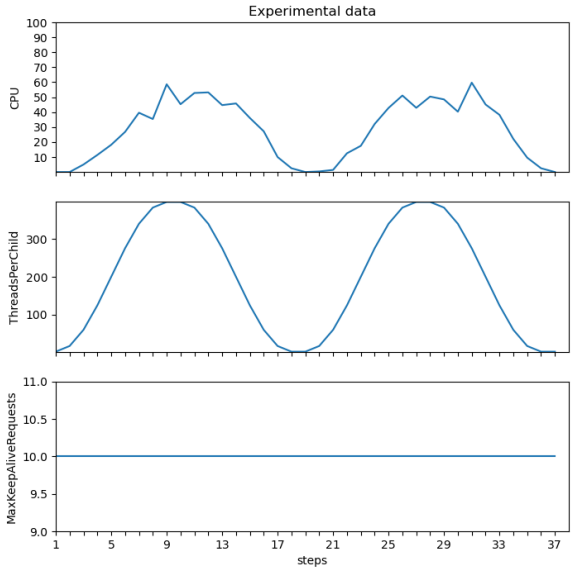

Рис. 16: Ход эксперимента 6.

ности построения модели в линейном приближении с достаточно высокой точностью, однако комбинация таких факторов и слишком большое их влияние на множество используемых значений управляющих параметров приводят к большим затруднениям с построением такой модели. В-четвертых, существенное значение имеют условия вариации параметров: длительность итерации, их число, способ усреднения. Кроме того, при выборе разных рабочих точек получаются совершенно различные коэффициенты уравнения динамики сервера.

В ходе проведения дополнительных экспериментов было установлено, что при увеличении длительности итерации происходит усреднение большего числа измеряемых значений загрузки ЦП, что позволяет нивелировать флуктуации и остаточное влияние перезапуска дочернего процесса сервера на использование им ресурсов процессора. К подобному результату приводит и использование медианы вместо арифметического среднего. Недостатком такого подхода является невозможность достаточно быстро корректировать работу сервера, поэтому при осуществлении реального управления выбор подходящей длительности итерации должен осуществляться как поиск компромисса [6].

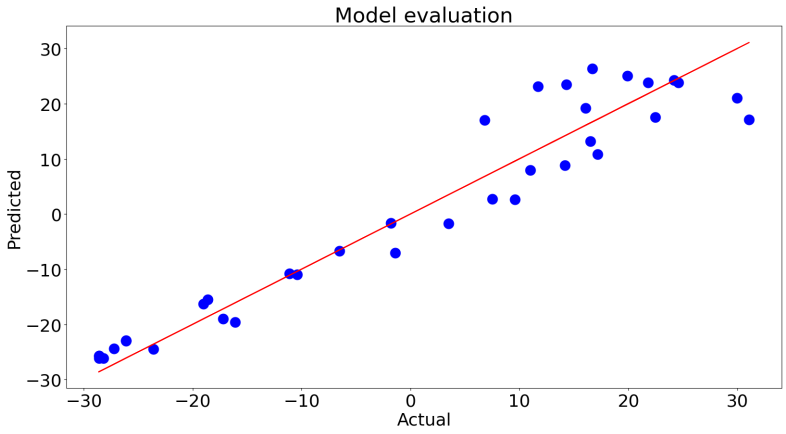

Рис. 17: Диаграмма точности модели, построенной в результате эксперимента 6.

В ряде сценариев экспериментов (2, 6, 7) вводилось время простаивания после перезапуска сервера, что позволило получать более похожие друг на друга результаты при проведении независимых экспериментов с одинаковыми параметрами. Тем не менее количество итераций оказывает заметное влияние на получаемые результаты. Для увеличения точности построения моделей необходимо увеличивать как число итераций, так и число синусоидальных волн. В случае неудачного выбора настроек итераций может быть получено уравнение динамики сервера, соотношение коэффициентов которого не отвечает предположениям о реальном характере динамики. Метрики R^2 и $RMSE$ будут при этом указывать на то, что модель обладает высокой точностью. Из этого можно сделать вывод, что вышеописанные метрики не отражают в полной мере качественных характеристик модели.

Следует отметить, что для построения управления в контуре обратной связи знание точных коэффициентов уравнения (1) не является обязательным, что позволяет избежать необходимости поиска оптимальных в строгом смысле условий проведения экспериментов и нивелировать многие из вышеописанных недостатков.

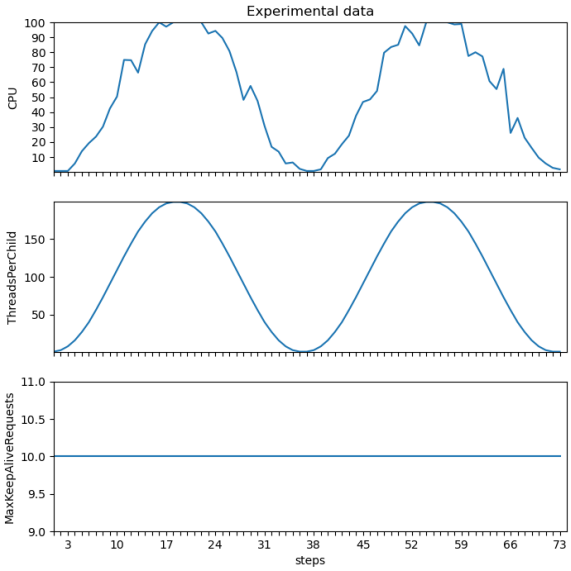

Рис. 18: Ход эксперимента 7.

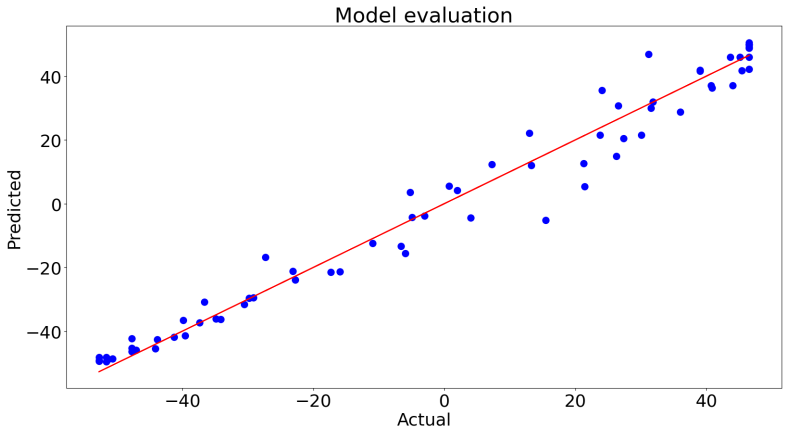

Рис. 19: Диаграмма точности модели, построенной в результате эксперимента 7.

Выводы

В настоящей работе рассмотрена динамика Apache HTTP Server, функционирующего под нагрузкой, созданной нагрузочным имитатором Apache JMeter. Исходная задача, основанная на представленных в [3, 2] методах, переработана с учетом современных реалий и практических ограничений. Выбраны управляющие параметры сервера. Показана возможность автоматизации проведения экспериментов, построения модели, а также синтеза управ-

ления по экспериментальным данным. Исследованы вопросы оценки точности и адекватности моделей. Несмотря на возникшие трудности в реализации, продемонстрированы эффективность и обоснованность предложенного подхода.

В дальнейшем предполагается совершенствование как условий нагрузочного тестирования, так и условий работы сервера в целях приближения их к реальным практикам, применяемым при организации функционирования современных веб-серверов. В частности, не исключено использование ОС Linux для сервера и применение усложненных сценариев нагрузочного тестирования. Кроме того, предполагается проведение дополнительных экспериментов, рассмотрение усовершенствованных моделей работы сервера и исследование различных типов регуляторов.

Заключение

На основе исследований, проведенных в рамках данной работы, получены следующие результаты, которые выносятся на защиту:

1. Организовано проведение натурных экспериментов с веб-сервером, исследован вопрос о выборе условий их реализации.

2. Рассмотрена задача о формировании модели динамики веб-сервера на основании экспериментальных данных с построением пропорционального регулятора на ее основе. Выполнены необходимые практические расчеты для рассматриваемого сервера.

3. Разработан программный продукт, с помощью которого реализуется автоматизированное проведение экспериментов и построение моделей.

Приложение А

Основные методы класса Server:

```
def __init__(self, config_path='C:/Apache24/conf', verbose=False):
        self._mpm_path = f'{config_path}/extra/httpd-mpm.conf'
        self._default_path = f'{config_path}/extra/httpd-default.conf'
        self.restart_needed = False

        self._tpc_regex = re.compile(r'(?<=>\n\tThreadsPerChild\s)\s*\d{1,4}', re.S)
        self._mkar_regex = re.compile(r'(?<=MaxKeepAliveRequests\s)\s*\d{1,4}')
        self.scan_configs()
        #psutils' Process objects
        self.parent_proc = None
        self.child_proc = None
        for process in pu.process_iter(['name']):
            if process.info['name'] == 'httpd.exe' and process.children():
                self.parent_proc = process
                self.child_proc = process.children()[0]
                break
        if not self.parent_proc or not self.child_proc:
            self.started = False
        else:
            self.started = True
        self.verbose = verbose


    def start(self):
        if not self.started:
            os.system('httpd.exe -k start')
            self.started = True
            for process in pu.process_iter(['name']):
                if process.info['name'] == 'httpd.exe' and process.children():
                    self.parent_proc = process
                    self.child_proc = process.children()[0]
                    break
            if not self.parent_proc or not self.child_proc:
                print('Failed to start server!')
                self.started = False
        if self.verbose:
            print('Server successfully started')
        return



    def restart(self):
        if not self.started:
            if self.verbose:
                print('Server is stopped. Starting...')
            self.start()
            return
        old_child_proc = self.child_proc
        os.system('httpd.exe -k restart')
        if self.restart_needed:
            self.restart_needed = False
        self.child_proc = self.parent_proc.children()[-1]
        if self.child_proc.pid == old_child_proc:
            print('Failed to restart server!')
            return
        if self.verbose:
            print('Server successfully restarted')
        return

    def stop(self):
        if self.started:
            os.system('httpd.exe -k stop')
            self.started = False
            if self.parent_proc.pid in pu.pids():
                print('Failed to stop server!')
```

```
                return
            else:
                if self.verbose:
                    print('Server successfully stopped')
                if not self.started:
                    self.parent_proc = None
                    self.child_proc = None
            else:
                if self.verbose:
                    print('Server already stopped')
            return
```

Вспомогательная функция `sine_wave`:

```
def sine_wave(mid, amplitude, steps):
    if steps % 2 != 0:
        steps -= 1
    x = np.linspace(0, np.pi, steps//2)
    x = np.append(x, np.linspace(np.pi, np.pi*2, steps//2))
    vals = mid - amplitude*np.cos(x)
    vals = np.round_(vals)
    return vals
```

Реализация автоматизированного проведения натурных экспериментов (опущены вывод графиков и проверка правильности входных аргументов):

```
def get_cpu_usage(self, interval=1.0, samples=10, median=False,
                                _logical=False, _allow_over_hundred=False):
        cpu_usage_data = np.zeros(samples)
        for i in range(samples+1):
            cur_cpu_usage = self.child_proc.cpu_percent(interval=0)
            cur_cpu_usage /= pu.cpu_count(logical=_logical)
            time.sleep(interval)
            if i > 0:
                cpu_usage_data[i-1] = cur_cpu_usage

                if self.verbose:
                    # print(f'CPU USAGE: {cur_cpu_usage}')
                    pass
        if not median:

            if _allow_over_hundred:
                return  '{:.{}f}'.format(np.mean(cpu_usage_data), 1)
            else:
                return '{:.{}f}'.format(min(np.mean(cpu_usage_data), 100.0), 1)
        elif not _allow_over_hundred:
            return '{:.{}f}'.format(np.median(cpu_usage_data), 1)
        else:
            return '{:.{}f}'.format(min(np.median(cpu_usage_data), 100.0), 1)


def vary_config(self, miso=False, primary_param='tpc', primary_mid=100,
                    primary_amplitude=99, primary_steps=120, primary_waves_count=1,
                    secondary_default_value=None, secondary_mid=8, secondary_amplitude=6,
                    secondary_waves_count=None, waves_number_offset=1.25,
                    step_time=10, sleep_time=3, interval=1.0,  median=False,
                    silent=False, csv_file=filename, skip_affirm=False):



    primary = Param_variable(primary_mid, primary_amplitude, primary_param,
                primary_steps, primary_waves_count)

    secondary_param = 'mkar' if primary_param == 'tpc' else 'tpc'

    if secondary_default_value is None:
```

```
    if secondary_param == 'mkar':
        secondary_default_value = 10
    else:
        secondary_default_value = 100

if not miso:
    secondary_mid = secondary_default_value
    secondary_amplitude = 0
    secondary_steps = primary_steps
    secondary_waves_count = primary_waves_count
    secondary = Param_variable(secondary_mid, secondary_amplitude, secondary_param,
                secondary_steps, secondary_waves_count)
else:
    secondary = Param_variable(secondary_mid, secondary_amplitude, secondary_param)
    secondary.align_with_primary(primary, waves_number_offset, secondary_waves_count)


if not silent:
    plt.ion()
    _, (ax1, ax2, ax3) = plt.subplots(3, 1, sharex=True, figsize=(8, 8))
    set_plot()
    plt.draw()
    plt.pause(0.001)


step_full_time = step_time + sleep_time
print(f'The test will take up to {step_full_time * primary.total_steps} seconds.')
if not skip_affirm:
    print(f'Type "Y" to proceed, "N" to cancel.')
    while True:
        sym = input()
        if sym in ['Y', 'y']:
            break
        if sym in ['N', 'n']:
            return


if not miso:
    info_string = f'with {secondary.name} constant at {secondary.mid}'
else:
    info_string = f'and {secondary.name} from {secondary.low} '
                          f'to {secondary.high} in {secondary.steps}'
    info_string += f' steps {secondary.waves_count} time'
                           f'{"s" if secondary.waves_count > 1 else ""}'
print(f'Starting sine wave benchmark for {primary.name} from '
         f'{primary.low} to {primary.high} in {primary.steps} steps'
        f' {primary.waves_count} time{"s" if primary.waves_count > 1 else ""} '
        f'{info_string} with me{"di"*median}an cpu usage count method')

total_steps = primary.total_steps + 1
samples = ceil(step_time / interval)
#config_range = np.tile(sine_wave(primary_mid, primary_amplitude, primary_steps), waves_count)
if self.verbose:
    print(f'Range of {primary.name} values: {primary.config_range}')
    if miso:
        print(f'Range of {secondary.name} values: {secondary.config_range}')
cpu_usage_data = np.zeros(total_steps)
time_range = np.arange(1, total_steps+1)
if primary.name == 'tpc':
    self.tpc = primary.low
    self.mkar = secondary.low
else:
    self.mkar = primary.low
    self.tpc = secondary.low
if self.restart_needed:
    self.restart()


for i in range(total_steps):
```

```
            cur_cpu_usage = self.get_cpu_usage(interval, samples, median)
            cpu_usage_data[i] = cur_cpu_usage
            cur_prim = primary.config_range[i]
            cur_sec = secondary.config_range[i]
            if primary_param == 'tpc':
                self.tpc = cur_prim
                self.mkar = cur_sec
            else:
                self.mkar = cur_prim
                self.tpc = cur_sec
            if self.verbose:
                if not miso:
                    secondary_info_string = ''
                else:
                    secondary_info_string = f'current {secondary.name}: {cur_sec},\n'
                print(f'\nCurrent {primary.name}: {cur_prim}, {secondary_info_string}'
                      f'current CPU usage: {cur_cpu_usage}, '
                      f'steps remaining: {total_steps - i - 1}')

            if self.restart_needed:
                self.restart()
            set_plot(i)
            plot_data()
            time.sleep(sleep_time)
            clear_plot()

        write_in_csv()
        #if self.verbose:
        print('Benchmarking done!')
        return
```

Реализация автоматизированного построения модели по экспериментальным данным (опущен вывод графика):

```
def read_csv(param='tpc', filename=csv_file):
    with open(filename, 'r') as file:
        reader = csv.reader(file, delimiter=',')
        next(reader, None)
        csv_data = np.array([[float(val) for val in row] for row in reader])
    if param == 'tpc':
        test_data = csv_data[:, 2::2]
    elif param == 'mkar':
        test_data = csv_data[:, 3:]
    elif param == 'miso':
        test_data = csv_data[:, 2:]
    else:
        print('Invalid argument!')
        return None, None

    u = test_data[:, :-1]
    y = test_data[:, -1:]
    mu = np.mean(u[:-1], axis=0)
    my = np.mean(y[1:])
    u = u - mu
    y = y - my
    return u, y, mu, my

def least_squares_arma(u, y):
    A = np.hstack((u[:-1], y[:-1]))
    coefs = np.linalg.lstsq(A, y[1:], rcond=None)[0].T
    return coefs[0]

def fit(u, y, coefs):
    _, ax = plt.subplots(figsize=(12, 12))
    y_pred = np.zeros(y.shape[0]-1)
    for i in range(coefs.shape[0]-1):
        y_pred += coefs[i] * u[:-1, i]
    y_pred += coefs[-1] * y[:-1, 0]
```

```
    ax.plot(y[1:], y_pred, 'bo', y, y, 'r-', ms=10)
    ax.set_title('Model evaluation')
    ax.set_xlabel('Actual')
    ax.set_ylabel('Predicted')
    return y_pred

def r_squared(y, y_pred):
    y_pred = y_pred.reshape((-1, 1))
    return 1 - (np.var(y[1:] - y_pred) / np.var(y[1:]))

def rmse(y, y_pred):
    y_pred = y_pred.reshape((-1, 1))
    residuals = np.square(y[1:] - y_pred)
    mean_val = np.mean(residuals)
    return np.sqrt(mean_val)

def print_equation(mu, my, coefs, param='tpc'):
    if isinstance(mu, float):
        mu = [mu]
    mu = [str(int(val)) for val in mu]
    opstring = ', '.join(mu)
    print(f'For {param} operating point: u = ({opstring}), y = ' + '{:.{}f}:'.format(my, 3))
    signs = []
    for i, coef in enumerate(coefs):
        if coef < 0:
            coefs[i] = -1*coefs[i]
            signs.append('-')
        else:
            signs.append('+' if i else '')

    if param == 'tpc':
        ustring = f'{signs[0]}' + '{:.{}f}*TPC'.format(coefs[0], 3)
    elif param == 'mkar':
        ustring = f'{signs[-2]}' + '{:.{}f}*MKAR'.format(coefs[-2], 3)
    elif param == 'miso':
        ustring = f'{signs[0]}' + '{:.{}f}*TPC'.format(coefs[0], 3)
        ustring += f' {signs[1]} ' +  '{:.{}f}*MKAR'.format(coefs[1], 3)
    else:
        print('Invalid argument!')
        return

    print(f'y(k+1) = {ustring} {signs[-1]} ' + '{:.{}f}*y(k)'.format(coefs[-1], 3))
    return
```

Список литературы

[1] Веремей Е. И. Линейные системы с обратной связью: Учебное пособие. СПб.: Лань, 2013. 448 с.

[2] Diao Y., Gandhi N., Hellerstein J. L., Parekh S., Tilbury D. M. Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache Web server // NOMS 2002. IEEE/IFIP Network Operations and Management Symposium. 2002. P. 219–234.

[3] Hellerstein J. L., Diao Y., Parekh S., Tilbury D. M. Feedback Control of Computing Systems. Hoboken: Wiley, 2004. 429 p.

[4] Liu Z., Niclausse N., Jalpa-Villanueva C. Traffic model and performance evaluation of Web servers // Performance Evaluation. 2001. Vol. 46, P. 77–100.

[5] Mosberger D., Jin T. «httperf: A tool for measuring web server performance,»// First Workshop on Internet Server Performance (WISP 98). 1998. P. 59–67.

[6] Philipp K. Janert. Feedback Control for Computer Systems. Sebastopol: O'Reilly, 2013. 336 p.

[7] Библиотека Numpy, официальный сайт [Электронный ресурс]: `https://numpy.org/` (дата обращения: 30.05.2020).

[8] Библиотека psutils, официальный сайт [Электронный ресурс]: `https://psutil.readthedocs.io/en/latest/` (дата обращения: 30.05.2020).

[9] Нагрузочный имитатор ApacheBench, раздел сайта Apache HTTP Server [Электронный ресурс]: `https://httpd.apache.org/docs/2.4/programs/ab.html` (дата обращения: 30.05.2020).

[10] Нагрузочный имитатор Apache JMeter, официальный сайт [Электронный ресурс]: `URL:https://jmeter.apache.org/` (дата обращения: 30.05.2020).

[11] Нагрузочный имитатор Gatling, официальный сайт [Электронный ресурс]: `https://gatling.io/` (дата обращения: 30.05.2020).

[12] Нагрузочный имитатор Goad, официальный сайт [Электронный ресурс]: https://goad.io/ (дата обращения: 30.05.2020).

[13] Нагрузочный имитатор Grinder, официальный сайт [Электронный ресурс]: http://grinder.sourceforge.net/ (дата обращения: 30.05.2020).

[14] Нагрузочный имитатор HTTPerf, официальный GitHub-репозиторий [Электронный ресурс]: https://github.com/httperf/httperf (дата обращения: 30.05.2020).

[15] Нагрузочный имитатор LoadImpact K6, официальный сайт [Электронный ресурс]: https://k6.io/ (дата обращения: 30.05.2020).

[16] Нагрузочный имитатор Locust, официальный сайт [Электронный ресурс]: https://locust.io/ (дата обращения: 30.05.2020).

[17] Нагрузочный имитатор OpenSTA, официальный сайт [Электронный ресурс]: http://www.opensta.org/ (дата обращения: 30.05.2020).

[18] Нагрузочный имитатор Tsung, официальный сайт [Электронный ресурс]: http://tsung.erlang-projects.org/ (дата обращения: 30.05.2020).

[19] Язык Python, официальный сайт [Электронный ресурс]: https://www.python.org/ (дата обращения: 30.05.2020).

[20] The Apache HTTP Server Project [Электронный ресурс]: URL:https://httpd.apache.org/ (дата обращения: 30.05.2020).

[21] Open Source Load Testing Tool Review 2020 [Электронный ресурс]: https://k6.io/blog/comparing-best-open-source-load-testing-tools (дата обращения: 30.05.2020).

[22] Ultimate Guide: 23 Free Load Testing Tools Reviewed [Электронный ресурс]: https://octoperf.com/blog/2017/11/21/open-source-load-testing-tools/ (дата обращения: 30.05.2020).